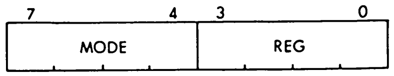
digital

VAX11 PROGRAMMING CARD

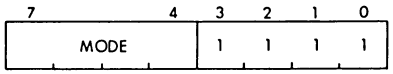
GENERAL REGISTER ADDRESSING MODES:



Operand Specifier Byte

				Access							
Hex	**Dec**	**Mode**	**Assembler**	**r**	**m**	**w**	**a**	**v**	**PC**	**SP**	**Indexable?**
0-3	0-3	literal	S^literal	y	f	f	f	f	—	—	f
4	4	indexed	i[Rx]	y	y	y	y	y	f	y	f
5	5	register	Rn	y	y	y	f	y	u	ug	f
6	6	register deferred	(Rn)	y	y	y	y	y	u	y	y
7	7	autodecrement	—(Rn)	y	y	y	y	y	u	y	ux
8	8	autoincrement	(Rn)+	y	y	y	y	y	p	y	ux
9	9	autoincr deferred	@(R)+	y	y	y	y	y	p	y	ux
A	10	byte displacement	B^D(Rn)	y	y	y	y	y	p	y	y
B	11	byte displ deferred	@B^D(Rn)	y	y	y	y	y	p	y	y
C	12	word displacement	W^D(Rn)	y	y	y	y	y	p	y	y
D	13	word displ deferred	@W^D(Rn)	y	y	y	y	y	p	y	y
E	14	longword displacement	L^D(Rn)	y	y	y	y	y	p	y	y
F	15	longword displ deferred	@L^D(Rn)	y	y	y	y	y	p	y	y

PROGRAM COUNTER ADDRESSING (reg = 15)



Operand Specifier Byte

				Access							
Hex	**Dec**	**Mode**	**Assembler**	**r**	**m**	**w**	**a**	**v**	**PC**	**SP**	**Indexable?**
8	8	immediate	I^constant	y	u	u	y	y	—	—	y
9	9	absolute	@address	y	y	y	y	y	—	—	y
A	10	byte relative	B^address	y	y	y	y	y	—	—	y
B	11	byte rel deferred	@B^address	y	y	y	y	y	—	—	y
C	12	word relative	W^address	y	y	y	y	y	—	—	y
D	13	word rel deferred	@W^address	y	y	y	y	y	—	—	y
E	14	longword relative	L^address	y	y	y	y	y	—	—	y
F	15	longword rel deferred	@L^address	y	y	y	y	y	—	—	y

ADDRESSING LEGEND

Access

r =	read
m =	modify
w =	write
a =	address
v =	field

Syntax

i =	any indexable addr mode
D =	displacement
Rn =	general register, n = 0-15
Rx =	general register, n = 0-14

Result

y =	yes, always valid addr mode
f =	reserved addr mode fault
— =	logically impossible
P =	Program Counter addressing
u =	unpredictable
uq =	unpredictable for quad and double (and field if position + size > 32)
ux =	unpredictable for index register same as base register

OPERAND SPECIFIER NOTATION LEGEND

The standard notation for operand specifiers is:

<name>.<access type><data type>

where:

1. Name is a suggestive name for the operand in the context of the instruction. It is the capitalized name of a register or block for implied operands.
2. Access type is a letter denoting the operand specifier access type.
   - a—Calculate the effective address of the specified operand. Address is returned in a pointer which is the actual instruction operand. Context of address calculation is given by data type given by <data type>.
   - b—No operand reference. Operand specifier is branch displacement. Size of branch displacement is given by <data type>.
   - m—operand is modified (both read and written)
   - r—operand is read only
   - v—if not "Rn", same as a. If "RN", R[n + 1]'R[n].
   - w—operand is written only
3. Data type is a letter denoting the data type of the operand
   - b—byte
   - d—double
   - f—floating
   - l—longword
   - q—quadword
   - v—field (used only on implied operands)
   - w—word
   - x—first data type specified by instruction
   - y—second data type specified by instruction
   - *—multiple longwords (used only on implied operands)
4. Implied operands, that is, locations that are accessed by the instruction, but not specified in an operand, are denoted in enclosing brackets, [ ].

CONDITION CODES LEGEND

* =	conditionally cleared/set
— =	not affected
0 =	cleared
1 =	set

INSTRUCTION SET

OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES N	Z	V	C
9D	ACBB	Add compare and branch byte	limit.rb, add.rb, index.mb, displ.bw	•	•	•	-
6F	ACBD	Add compare and branch double	limit.rd, add.rd, index.md, displ.bw	•	•	•	-
4F	ACBF	Add compare and branch floating	limit.rf, add.rf, index.mf, displ.bw	•	•	•	-
F1	ACBL	Add compare and branch long	limit.rl, add.rl, index.ml, displ.bw	•	•	•	-
3D	ACBW	Add compare and branch word	limit.rw, add.rw, index.mw, displ.bw	•	•	•	-
58	ADAWI	Add aligned word interlocked	add.rw, sum.mw	•	•	•	•
80	ADDB2	Add byte 2-operand	add.rb, sum.mb	•	•	•	•
81	ADDB3	Add byte 3-operand	add1.rb, add2.rb, sum.wb	•	•	•	0
60	ADDD2	Add double 2-operand	add.rd, sum.md	•	•	•	0
61	ADDD3	Add double 3-operand	add1.rd, add2.rd, sum.wd	•	•	•	0
40	ADDF2	Add floating 2-operand	add.rf, sum.mf	•	•	•	0
41	ADDF3	Add floating 3-operand	add1.rf, add2.rf, sum.wf	•	•	•	0
C0	ADDL2	Add long 2-operand	add.rl, sum.ml	•	•	•	•
C1	ADDL3	Add long 3-operand	add1.rl, add2.rl, sum.wl	•	•	•	0
20	ADDP4	Add packed 4-operand	addlen.rw, addaddr.ab, sumlen.rw, sumaddr.ab, [R0-3.wl]	•	•	•	0
21	ADDP6	Add packed 6-operand	add1len.rw, add1addr.ab, add2len.rw, add2addr.ab, sumlen.rw, sumaddr.ab, [R0-5.wl]	•	•	•	0
A0	ADDW2	Add word 2-operand	add.rw, sum.mw	•	•	•	•
A1	ADDW3	Add word 3-operand	add1.rw, add2.rw, sum.ww	•	•	•	0
D8	ADWC	Add with carry	add.rl, sum.ml	•	•	•	•
F3	AOBLEQ	Add one and branch on less or equal	limit.rl, index.ml, displ.bb	•	•	•	-
F2	AOBLSS	Add one and branch on less	limit.rl, index.ml, displ.bb	•	•	•	-
78	ASHL	Arithmetic shift long	count.rb, src.rl, dst.wl	•	•	•	0
F8	ASHP	Arithmetic shift and round packed	count.rb, srclen.rw, srcaddr.ab, round.rb, dstlen.rw, dstaddr.ab, [R0-3.wl]	•	•	•	0
79	ASHQ	Arithmetic shift quad	count.rb, src.rq, dst.wq	•	•	•	0
E1	BBC	Branch on bit clear	pos.rl, base.vb, displ.bb, [field.rv]	-	-	-	-
E5	BBCC	Branch on bit clear and clear	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E7	BBCCI	Branch on bit clear and clear interlocked	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E3	BBCS	Branch on bit clear and set	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E0	BBS	Branch on bit set	pos.rl, base.vb, displ.bb, [field.rv]	-	-	-	-
E4	BBSC	Branch on bit set and clear	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E2	BBSS	Branch on bit set and set	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
E6	BBSSI	Branch on bit set and set interlocked	pos.rl, base.vb, displ.bb, [field.mv]	-	-	-	-
1E	BCC	Branch on carry clear	displ.bb	-	-	-	-
1F	BCS	Branch on carry set	displ.bb	-	-	-	-
13	BEQL	Branch on equal	displ.bb	-	-	-	-
13	BEQLU	Branch on equal unsigned	displ.bb	-	-	-	-
18	BGEQ	Branch on greater or equal	displ.bb	-	-	-	-
1E	BGEQU	Branch on greater or equal unsigned	displ.bb	-	-	-	-
14	BGTR	Branch on greater	displ.bb	-	-	-	-
1A	BGTRU	Branch on greater unsigned	displ.bb	-	-	-	-

OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES N	Z	V	C
8A	BICB2	Bit clear byte 2-operand	mask.rb, dst.mb	•	•	0	-
8B	BICB3	Bit clear byte 3-operand	mask.rb,src.rb, dst.wb	•	•	0	-
CA	BICL2	Bit clear long 2-operand	mask.rl, dst.ml	•	•	0	-
CB	BICL3	Bit clear long 3-operand	mask.rl, src.rl, dst.wl	•	•	0	-
B9	BICPSW	Bit clear processor status word	mask.rw	•	•	•	•
AA	BICW2	Bit clear word 2-operand	mask.rw,dst.mw	•	•	0	-
AB	BICW3	Bit clear word 3-operand	mask.rw, src.rw, dst.ww	•	•	0	-
88	BISB2	Bit set byte 2-operand	mask.rb, dst.mb	•	•	0	-
89	BISB3	Bit set byte 3-operand	mask.rb, src.rb, dst.wb	•	•	0	-
C8	BISL2	Bit set long 2-operand	mask.rl, dst.ml	•	•	0	-
C9	BISL3	Bit set long 3-operand	mask.rl, src.rl, dst.wl	•	•	0	-
B8	BISPSW	Bit set processor status word	mask.rw	•	•	•	•
A8	BISW2	Bit set word 2-operand	mask.rw,dst.mw	•	•	0	-
A9	BISW3	Bit set word 3-operand	mask.rw, src.rw, dst.ww	•	•	0	-
93	BITB	Bit test byte	mask.rb, src.rb	•	•	0	-
D3	BITL	Bit test long	mask.rl, src.rl	•	•	0	-
B3	BITW	Bit test word	mask.rw, src.rw	•	•	0	-
E9	BLBC	Branch on low bit clear	src.rl, displ.bb	-	-	-	-
E8	BLBS	Branch on low bit set	src.rl, displ.bb	-	-	-	-
15	BLEQ	Branch on less or equal	displ.bb	-	-	-	-
1B	BLEQU	Branch on less or equal unsigned	displ.bb	-	-	-	-
19	BLSS	Branch on less	displ.bb	-	-	-	-
1F	BLSSU	Branch on less unsigned	displ.bb	-	-	-	-
12	BNEQ	Branch on not equal	displ.bb	-	-	-	-
12	BNEQU	Branch on not equal unsigned	displ.bb	-	-	-	-
03	BPT	Break point fault	[−(KSP).w•]	0	0	0	0
11	BRB	Branch with byte displacement	displ.bb	-	-	-	-
31	BRW	Branch with word displacement	displ.bw	-	-	-	-
10	BSBB	Branch to subroutine with byte displacement	displ.bb, [−(SP).wl]	-	-	-	-
30	BSBW	Branch to subroutine with word displacement	displ.bw, [−(SP).wl]	-	-	-	-
1C	BVC	Branch on overflow clear	displ.bb	-	-	-	-
1D	BVS	Branch on overflow set	displ.bb	-	-	-	-
FA	CALLG	Call with general argument list	arglist.ab,dst.ab, [−(SP).w•]	0	0	0	0
FB	CALLS	Call with argument list on stack	numarg.rl,dst.ab, [−(SP).w•]	0	0	0	0
8F	CASEB	Case byte	selector.rb, base.rb, limit.rb, displ.bw-list	•	•	0	•
CF	CASEL	Case long	selector.rl, base.rl, limit.rl, displ.bw-list	•	•	0	•
AF	CASEW	Case word	selector.rw, base.rw, limit.rw, displ.bw-list	•	•	0	•
BD	CHME	Change mode to executive	param.rw, [−(ySP).w•] y = MINU(E, PSL*current-mode*)	0	0	0	0
BC	CHMK	Change mode to kernal	param.rw, [−(KSP).w•]	0	0	0	0
BE	CHMS	Change mode to supervisor	param.rw, [−(ySP).w•] y = MINU(S, PSL*current-mode*)	0	0	0	0
BF	CHMU	Change mode to user	param.rw, [−(SP).w•]	0	0	0	0
94	CLRB	Clear byte	dst.wb	0	1	0	-
7C	CLRD	Clear double	dst.wd	0	1	0	-

OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES N	Z	V	C
D4	CLRF	Clear floating	dst.wf	0	1	0	-
D4	CLRL	Clear long	dst.wl	0	1	0	-
7C	CLRQ	Clear quad	dst.wq	0	1	0	-
B4	CLRW	Clear word	dst.ww	0	1	0	-
91	CMPB	Compare byte	src1.rb, src2.rb	•	•	0	•
29	CMPC3	Compare character 3-operand	len.rw, src1addr.ab, src2addr.ab, [R0-3.wl]	•	•	0	•
2D	CMPC5	Compare character 5-operand	src1len.rw, src1addr.ab, fill.rb, src2len.rw, src2addr.ab, [R0-3.wl]	•	•	0	•
71	CMPD	Compare double	src1.rd, src2.rd	•	•	0	0
51	CMPF	Compare floating	src1.rf, src2.rf	•	•	0	0
D1	CMPL	Compare long	src1.rl, src2.rl	•	•	0	•
35	CMPP3	Compare packed 3-operand	len.rw, src1addr.ab, src2addr.ab, [R0-3.wl]	•	•	0	0
37	CMPP4	Compare packed 4-operand	src1len.rw, src1addr.ab, src2len.rw, src2addr.ab, [R0-3.wl]	•	•	0	0
EC	CMPV	Compare field	pos.rl, size.rb, base.vb, [field.rv], src.rl	•	•	0	•
B1	CMPW	Compare word	src1.rw, src2.rw	•	•	0	•
ED	CMPZV	Compare zero-extended field	pos.rl, size.rb, base.vb, [field.rv], src.rl	•	•	0	•
0B	CRC	Calculate cyclic redundancy check	tbl.ab, initialcrc.rl, strlen.rw, stream.ab, [R0-5.wl], dst.wl	•	•	0	-
6C	CVTBD	Convert byte to double	src.rb, dst.wd	•	•	•	0
4C	CVTBF	Convert byte to floating	src.rb, dst.wf	•	•	•	0
98	CVTBL	Convert byte to long	src.rb, dst.wl	•	•	•	0
99	CVTBW	Convert byte to word	src.rb, dst.ww	•	•	•	0
68	CVTDB	Convert double to byte	src.rd, dst.wb	•	•	•	0
76	CVTDF	Convert double to floating	src.rd, dst.wf	•	•	•	0
6A	CVTDL	Convert double to long	src.rd, dst.wl	•	•	•	0
69	CVTDW	Convert double to word	src.rd, dst.ww	•	•	•	0
48	CVTFB	Convert floating to byte	src.rf, dst.wb	•	•	•	0
56	CVTFD	Convert floating to double	src.rf, dst.wd	•	•	•	0
4A	CVTFL	Convert floating to long	src.rf, dst.wl	•	•	•	0
49	CVTFW	Convert floating to word	src.rf, dst.ww	•	•	•	0
F6	CVTLB	Convert long to byte	src.rl, dst.wb	•	•	•	0
6E	CVTLD	Convert long to double	src.rl, dst.wd	•	•	•	0
4E	CVTLF	Convert long to floating	src.rl, dst.wf	•	•	•	0
F9	CVTLP	Convert long to packed	src.rl, dstlen.rw, dstaddr.ab, [R0-3.wl]	•	•	•	0
F7	CVTLW	Convert long to word	src.rl, dst.ww	•	•	•	0
36	CVTPL	Convert packed to long	srclen.rw, srcaddr.ab, [R0-3.wl], dst.wl	•	•	•	0
08	CVTPS	Convert packed to leading separate	srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	•	•	•	0
24	CVTPT	Convert packed to trailing	srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	•	•	•	0
6B	CVTRDL	Convert rounded double to long	src.rd, dst.wl	•	•	•	0
4B	CVTRFL	Convert rounded floating to long	src.rf, dst.wl	•	•	•	0
09	CVTSP	Convert leading separate to packed	srclen.rw, srcaddr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	•	•	•	0

OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES N	Z	V	C
26	CVTTP	Convert trailing to packed	srclen.rw, srcaddr.ab, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-3.wl]	•	•	•	0
33	CVTWB	Convert word to byte	src.rw, dst.wb	•	•	•	0
6D	CVTWD	Convert word to double	src.rw, dst.wd	•	•	•	0
4D	CVTWF	Convert word to floating	src.rw, dst.wf	•	•	•	0
32	CVTWL	Convert word to long	src.rw, dst.wl	•	•	•	0
97	DECB	Decrement byte	dif.mb	•	•	•	•
D7	DECL	Decrement long	dif.ml	•	•	•	•
B7	DECW	Decrement word	dif.mw	•	•	•	•
86	DIVB2	Divide byte 2-operand	divr.rb, quo.mb	•	•	•	0
87	DIVB3	Divide byte 3-operand	divr.rb, divd.rb, quo.wb	•	•	•	0
66	DIVD2	Divide double 2-operand	divr.rd, quo.md	•	•	•	0
67	DIVD3	Divide double 3-operand	divr.rd, divd.rd, quo.wd	•	•	•	0
46	DIVF2	Divide floating 2-operand	divr.rf, quo.mf	•	•	•	0
47	DIVF3	Divide floating 3-operand	divr.rf, divd.rf, quo.wf	•	•	•	0
C6	DIVL2	Divide long 2-operand	divr.rl, quo.ml	•	•	•	0
C7	DIVL3	Divide long 3-operand	divr.rl, divd.rl, quo.wl	•	•	•	0
27	DIVP	Divide packed	divrlen.rw, divraddr.ab, divdlen.rw, divdaddr.ab, quolen.rw, quoaddr.ab, [R0-5.wl, −16(SP):−1(SP).wb]	•	•	•	0
A6	DIVW2	Divide word 2-operand	divr.rw, quo.mw	•	•	•	0
A7	DIVW3	Divide word 3-operand	divr.rw, divd.rw, quo.ww	•	•	•	0
38	EDITPC	Edit packed to character string	srclen.rw, srcaddr.ab, pattern.ab, dstaddr.ab, [R0-5.wl]	•	•	•	•
7B	EDIV	Extended divide	divr.rl, divd.rq, quo.wl, rem.wl	•	•	•	0
74	EMODD	Extended modulus double	mulr.rd, mulrx.rb, muld.rd, int.wl, fract.wd	•	•	•	0
54	EMODF	Extended modulus floating	mulr.rf, mulrx.rb, muld.rf, int.wl fract.wf	•	•	•	0
7A	EMUL	Extended multiply	mulr.rl, muld.rl, add.rl, prod.wq	•	•	0	0
EE	EXTV	Extract field	pos.rl, size.rb, base.vb, [field.rv], dst.wl	•	•	0	-
EF	EXTZV	Extract zero-extended field	pos.rl, size.rb, base.vb, [field.rv], dst.wl	•	•	0	-
EB	FFC	Find first clear bit	startpos.rl, size.rb, base.vb, [field.rv], findpos.wl	0	•	0	0
EA	FFS	Find first set bit	startpos.rl, size.rb, base.vb, [field.rv], findpos.wl	0	•	0	0
00	HALT	Halt (Kernel Mode only)	[−(KSP).w*]	•	•	•	•
96	INCB	Increment byte	sum.mb	•	•	•	•
D6	INCL	Increment long	sum.ml	•	•	•	•
B6	INCW	Increment word	sum.mw	•	•	•	•
0A	INDEX	Index calculation	subscript.rl, low.rl, high.rl, size.rl, entry.rl, addr.wl	•	•	0	0
0E	INSQUE	Insert into queue	entry.ab, addr.wl	•	•	0	•
F0	INSV	Insert field	src.rl, pos.rl, size.rb, base.vb, [field.wv]	0	0	0	-
17	JMP	Jump	dst.ab	-	-	-	-
16	JSB	Jump to subroutine	dst.ab, [−(SP)+.wl]	-	-	-	-
06	LDPCTX	Load process context (only legal on interrupt stack)	[PCB.r*, −(KSP).w*]	-	-	-	-

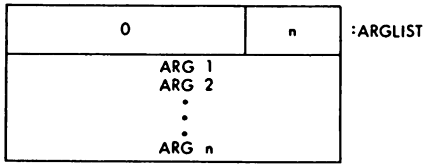
OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES N	Z	V	C
3A	LOCC	Locate character	char.rb, len.rw, addr.ab, [R0-1.wl]	0	•	0	0
39	MATCHC	Match characters	len1.rw, addr1.ab, len2.rw, addr2.ab, [R0-3.wl]	0	•	0	0
92	MCOMB	Move complemented byte	src.rb, dst.wb	•	•	0	-
D2	MCOML	Move complemented long	src.rl, dst.wl	•	•	0	-
B2	MCOMW	Move complemented word	src.rw, dst.ww	•	•	0	-
DB	MFPR	Move from processor register (Kernel Mode only)	procreg.rl, dst.wl	•	•	0	-
8E	MNEGB	Move negated byte	src.rb, dst.wb	•	•	•	•
72	MNEGD	Move negated double	src.rd, dst.wd	•	•	0	0
52	MNEGF	Move negated floating	src.rf, dst.wf	•	•	0	0
CE	MNEGL	Move negated long	src.rl, dst.wl	•	•	•	•
AE	MNEGW	Move negated word	src.rw, dst.ww	•	•	•	•
9E	MOVAB	Move address of byte	src.ab, dst.wl	•	•	0	-
7E	MOVAD	Move address of double	src.aq, dst.wl	•	•	0	-
DE	MOVAF	Move address of floating	src.al, dst.wl	•	•	0	-
DE	MOVAL	Move address of long	src.al, dst.wl	•	•	0	-
7E	MOVAQ	Move address of quad	src.aq, dst.wl	•	•	0	-
3E	MOVAW	Move address of word	src.aw, dst.wl	•	•	0	-
90	MOVB	Move byte	src.rb, dst.wb	•	•	0	-
28	MOVC3	Move character 3-operand	len.rw, srcaddr.ab, dstaddr.ab, [R0-5.wl]	0	1	0	0
2C	MOVC5	Move character 5-operand	srclen.rw, srcaddr.ab, fill.rb, dstlen.rw, dstaddr.ab, [R0-5.wl]	•	•	0	•
70	MOVD	Move double	src.rd, dst.wd	•	•	0	-
50	MOVF	Move floating	src.rf, dst.wf	•	•	0	-
D0	MOVL	Move long	src.rl, dst.wl	•	•	0	-
34	MOVP	Move packed	len.rw, srcaddr.ab, dstaddr.ab, [R0-3.wl]	•	•	0	-
DC	MOVPSL	Move processor status longword	dst.wl	-	-	-	-
7D	MOVQ	Move quad	src.rq, dst.wq	•	•	0	-
2E	MOVTC	Move translated characters	srclen.rw, srcaddr.ab, fill.rb, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-5.wl]	•	•	0	•
2F	MOVTUC	Move translated until character	srclen.rw, srcaddr.ab, escape.rb, tbladdr.ab, dstlen.rw, dstaddr.ab, [R0-5.wl]	•	•	•	•
B0	MOVW	Move word	src.rw, dst.ww	•	•	0	-
9A	MOVZBL	Move zero-extended byte to long	src.rb, dst.wl	0	•	0	-
9B	MOVZBW	Move zero-extended byte to word	src.rb, dst.ww	0	•	0	-
3C	MOVZWL	Move zero-extended word to long	src.rw, dst.wl	0	•	0	-
DA	MTPR	Move to processor register (Kernel Mode only)	src.rl, procreg.rl	•	•	0	-
84	MULB2	Multiply byte 2-operand	mulr.rb, prod.mb	•	•	•	0
85	MULB3	Multiply byte 3-operand	mulr.rb, muld.rb, prod.wb	•	•	•	0
64	MULD2	Multiply double 2-operand	mulr.rd, prod.md	•	•	•	0
65	MULD3	Multiply double 3-operand	mulr.rd, muld.rd, prod.wd	•	•	•	0
44	MULF2	Multiply floating 2-operand	mulr.rf, prod.mf	•	•	•	0
45	MULF3	Multiply floating 3-operand	mulr.rf, muld.rf, prod.wf	•	•	•	0
C4	MULL2	Multiply long 2-operand	mulr.rl, prod.ml	•	•	•	0

OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES			
				N	Z	V	C
C5	MULL3	Multiply long 3-operand	mulr.rl, muld.rl, prod.wl	•	•	•	0
25	MULP	Multiply packed	mulrlen.rw, mulradr.ab, muldlen.rw, muldadr.ab, prodlen.rw, prodadr.ab, [R0-5.wl]	•	•	•	0
A4	MULW2	Multiply word 2-operand	mulr.rw, prod.mw	•	•	•	0
A5	MULW3	Multiply word 3-operand	mulr.rw, muld.rw, prod.ww	•	•	•	0
01	NOP	No operation		-	-	-	-
75	POLYD	Evaluate polynomial double	arg.rd, degree.rw, tbladdr.ab, [R0-5.wl]	•	•	•	0
55	POLYF	Evaluate polynomial floating	arg.rf, degree.rw, tbladdr.ab, [R0-3.wl]	•	•	•	0
BA	POPR	Pop registers	mask.rw, [(SP)+.r*]	-	-	-	-
0C	PROBER	Probe read access	mode.rb, len.rw, base.ab	0	•	0	-
0D	PROBEW	Probe write access	mode.rb, len.rw, base.ab	0	•	0	-
9F	PUSHAB	Push address of byte	src.ab, [−(SP).wl]	•	•	0	-
7F	PUSHAD	Push address of double	src.aq, [−(SP).wl]	•	•	0	-
DF	PUSHAF	Push address of floating	src.al, [−(SP).wl]	•	•	0	-
DF	PUSHAL	Push address of long	src.al, [−(SP).wl]	•	•	0	-
7F	PUSHAQ	Push address of quad	src.aq, [−(SP).wl]	•	•	0	-
3F	PUSHAW	Push address of word	src.aw, [−(SP).wl]	•	•	0	-
DD	PUSHL	Push long	src.rl, [−(SP).wl]	•	•	0	-
BB	PUSHR	Push registers	mask.rw, [−(SP).w*]	-	-	-	-
02	REI	Return from exception or interrupt	[(SP)+.r*]	•	•	•	•
0F	REMQUE	Remove from queue	entry.ab, addr.wl	•	•	•	•
04	RET	Return from procedure	[(SP)+.r*]	•	•	•	•
9C	ROTL	Rotate long	count.rb, src.rl, dst.wl	•	•	0	-
05	RSB	Return from subroutine	[(SP)+.rl]	-	-	-	-
D9	SBWC	Subtract with carry	sub.rl, dif.ml	•	•	•	•
2A	SCANC	Scan for character	len.rw, addr.ab, tbladdr.ab, mask.rb, [R0-3.wl]	0	•	0	0
3B	SKPC	Skip character	char.rb, len.rw, addr.ab, [R0-1.wl]	0	•	0	0
F4	SOBGEQ	Subtract one and branch on greater or equal	index.ml, displ.bb	•	•	•	-
F5	SOBGTR	Subtract one and branch on greater	index.ml, displ.bb	•	•	•	-
2B	SPANC	Span characters	len.rw, addr.ab, tbladdr.ab, mask.rb, [R0-3.wl]	0	•	0	0
82	SUBB2	Subtract byte 2-operand	sub.rb, dif.mb	•	•	•	•
83	SUBB3	Subtract byte 3-operand	sub.rb, min.rb, dif.wb	•	•	•	0
62	SUBD2	Subtract double 2-operand	sub.rd, dif.md	•	•	•	0
63	SUBD3	Subtract double 3-operand	sub.rd, min.rd, dif.wd	•	•	•	0
42	SUBF2	Subtract floating 2-operand	sub.rf, dif.mf	•	•	•	0
43	SUBF3	Subtract floating 3-operand	sub.rf, min.rf, dif.wf	•	•	•	0
C2	SUBL2	Subtract long 2-operand	sub.rl, dif.ml	•	•	•	•
C3	SUBL3	Subtract long 3-operand	sub.rl, min.rl, dif.wl	•	•	•	0
22	SUBP4	Subtract packed 4-operand	sublen.rw, subaddr.ab, diflen.rw, difaddr.ab, [R0-3.wl]	•	•	•	0
23	SUBP6	Subtract packed 6 operand	sublen.rw, subaddr.ab, minlen.rw, minaddr.ab, diflen.rw, difaddr.ab, [R0-5.wl]	•	•	•	0
A2	SUBW2	Subtract word 2-operand	sub.rw, dif.mw	•	•	•	•

OP	MNEMONIC	DESCRIPTION	ARGUMENTS	COND. CODES N	Z	V	C
A3	SUBW3	Subtract word 3-operand	sub.rw, min.rw, dif.ww	•	•	•	0
07	SVPCTX	Save process context (Kernel Mode only)	[(SP)+.r•, −(KSP).w•]	-	-	-	-
95	TSTB	Test byte	src.rb	•	•	0	0
73	TSTD	Test double	src.rd	•	•	0	0
53	TSTF	Test floating	src.rf	•	•	0	0
D5	TSTL	Test long	src.rl	•	•	0	0
B5	TSTW	Test word	src.rw	•	•	0	0
FC	XFC	Extended function call	user defined operands	0	0	0	0
8C	XORB2	Exclusive OR byte 2-operand	mask.rb, dst.mb	•	•	0	-
8D	XORB3	Exclusive OR byte 3-operand	mask.rb, src.rb, dst.wb	•	•	0	-
CC	XORL2	Exclusive OR long 2-operand	mask.rl, dst.ml	•	•	0	-
CD	XORL3	Exclusive OR long 3-operand	mask.rl, src.rl, dst.wl	•	•	0	-
AC	XORW2	Exclusive OR word 2-operand	mask.rw, dst.mw	•	•	0	-
AD	XORW3	Exclusive OR word 3-operand	mask.rw, src.rw, dst.ww	•	•	0	-

CALL INSTRUCTION ARGUMENT LIST FORMAT

The format of an argument list is a sequence of longwords:



The argument count n is an unsigned byte contained in the first byte of the list. The high order 24 bits of the first longword are reserved for future use and must be zero. To access the argument count, the called procedure must ignore the reserved bits and pick up the count with the equivalent of a MOVZBL instruction.

INSTRUCTIONS

NUMERIC ORDER

00	HALT	40	ADDF2	80	ADDB2	C0	ADDL2
01	NOP	41	ADDF3	81	ADDB3	C1	ADDL3
02	REI	42	SUBF2	82	SUBB2	C2	SUBL2
03	BPT	43	SUBF3	83	SUBB3	C3	SUBL3
04	RET	44	MULF2	84	MULB2	C4	MULL2
05	RSB	45	MULF3	85	MULB3	C5	MULL3
06	LDPCTX	46	DIVF2	86	DIVB2	C6	DIVL2
07	SVPCTX	47	DIVF3	87	DIVB3	C7	DIVL3
08	CVTPS	48	CVTFB	88	BISB2	C8	BISL2
09	CVTSP	49	CVTFW	89	BISB3	C9	BISL3
0A	INDEX	4A	CVTFL	8A	BICB2	CA	BICL2
0B	CRC	4B	CVTRFL	8B	BICB3	CB	BICL3
0C	PROBER	4C	CVTBF	8C	XORB2	CC	XORL2
0D	PROBEW	4D	CVTWF	8D	XORB3	CD	XORL3
0E	INSQUE	4E	CVTLF	8E	MNEGB	CE	MNEGL
0F	REMQUE	4F	ACBF	8F	CASEB	CF	CASEL
10	BSBB	50	MOVF	90	MOVB	D0	MOVL
11	BRB	51	CMPF	91	CMPB	D1	CMPL
12	BNEQ, BNEQU	52	MNEGF	92	MCOMB	D2	MCOML
13	BEQL, BEQLU	53	TSTF	93	BITB	D3	BITL
14	BGTR	54	EMODF	94	CLRB	D4	CLRF, CLRL
15	BLEQ	55	POLYF	95	TSTB	D5	TSTL
16	JSB	56	CVTFD	96	INCB	D6	INCL
17	JMP	57	reserved	97	DECB	D7	DECL
18	BGEQ	58	ADAWI	98	CVTBL	D8	ADWC
19	BLSS	59	reserved	99	CVTBW	D9	SBWC
1A	BGTRU	5A	reserved	9A	MOVZBL	DA	MTPR
1B	BLEQU	5B	reserved	9B	MOVZBW	DB	MFPR
1C	BVC	5C	reserved	9C	ROTL	DC	MOVPSL
1D	BVS	5D	reserved	9D	ACBB	DD	PUSHL
1E	BCC, BGEQU	5E	reserved	9E	MOVAB	DE	MOVAF, MOVAL
1F	BCS, BLSSU	5F	reserved	9F	PUSHAB	DF	PUSHAF, PUSHA
20	ADDP4	60	ADDD2	A0	ADDW2	E0	BBS
21	ADDP6	61	ADDD3	A1	ADDW3	E1	BBC
22	SUBP4	62	SUBD2	A2	SUBW2	E2	BBSS
23	SUBP6	63	SUBD3	A3	SUBW3	E3	BBCS
24	CVTPT	64	MULD2	A4	MULW2	E4	BBSC
25	MULP	65	MULD3	A5	MULW3	E5	BBCC
26	CVTTP	66	DIVD2	A6	DIVW2	E6	BBSSI
27	DIVP	67	DIVD3	A7	DIVW3	E7	BBCCI
28	MOVC3	68	CVTDB	A8	BISW2	E8	BLBS
29	CMPC3	69	CVTDW	A9	BISW3	E9	BLBC
2A	SCANC	6A	CVTDL	AA	BICW2	EA	FFS
2B	SPANC	6B	CVTRDL	AB	BICW3	EB	FFC
2C	MOVC5	6C	CVTBD	AC	XORW2	EC	CMPV
2D	CMPC5	6D	CVTWD	AD	XORW3	ED	CMPZV
2E	MOVTC	6E	CVTLD	AE	MNEGW	EE	EXTV
2F	MOVTUC	6F	ACBD	AF	CASEW	EF	EXTZV
30	BSBW	70	MOVD	B0	MOVW	F0	INSV
31	BRW	71	CMPD	B1	CMPW	F1	ACBL
32	CVTWL	72	MNEGD	B2	MCOMW	F2	AOBLSS
33	CVTWB	73	TSTD	B3	BITW	F3	AOBLEQ
34	MOVP	74	EMODD	B4	CLRW	F4	SOBGEQ
35	CMPP3	75	POLYD	B5	TSTW	F5	SOBGTR
36	CVTPL	76	CVTDF	B6	INCW	F6	CVTLB
37	CMPP4	77	reserved	B7	DECW	F7	CVTLW
38	EDITPC	78	ASHL	B8	BISPSW	F8	ASHP
39	MATCHC	79	ASHQ	B9	BICPSW	F9	CVTLP
3A	LOCC	7A	EMUL	BA	POPR	FA	CALLG
3B	SKPC	7B	EDIV	BB	PUSHR	FB	CALLS
3C	MOVZWL	7C	CLRD, CLRQ	BC	CHMK	FC	XFC
3D	ACBW	7D	MOVQ	BD	CHME	FD	reserved
3E	MOVAW	7E	MOVAD, MOVAQ	BE	CHMS	FE	reserved
3F	PUSHAW	7F	PUSHAD, PUSHAQ	BF	CHMU	FF	reserved

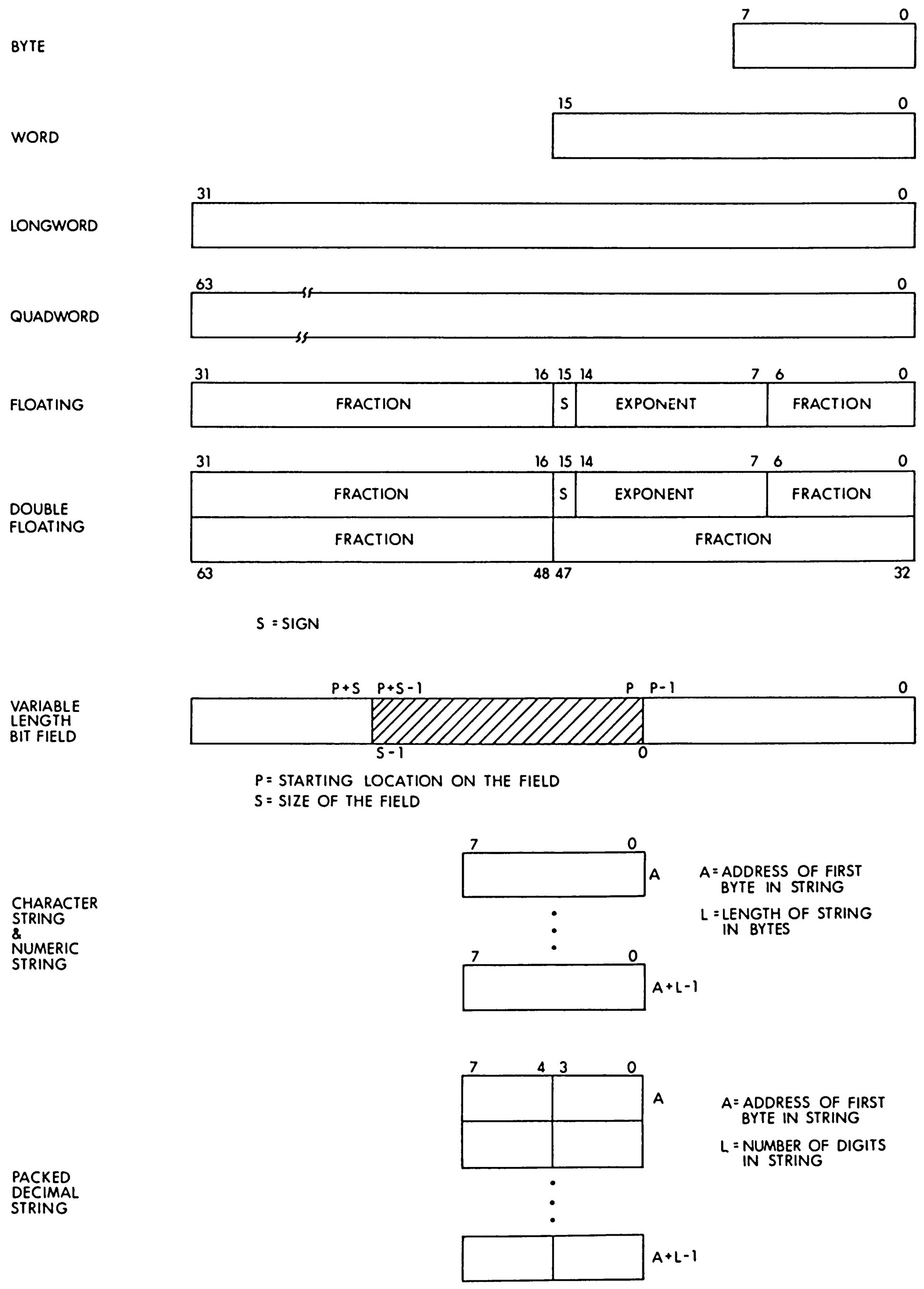
HEXADECIMAL—ASCII CONVERSION

HEX Code	ASCII Char	HEX Code	ASCII Char	HEX Code	ASCII Char	HEX Code	ASCII Char
00	NUL	20	SP	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	–	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

HEXADECIMAL TO DECIMAL CONVERSION TABLE

8		7		6		5		4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

DATA TYPES

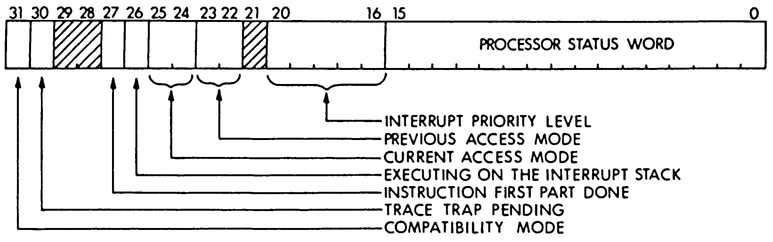


BYTE
7 0
WORD
15 0
LONGWORD
31 0
QUADWORD
63 0
FLOATING
31 16 15 14 7 6 0
FRACTION
S
EXPONENT
FRACTION
DOUBLE FLOATING
31 16 15 14 7 6 0
FRACTION
S
EXPONENT
FRACTION
FRACTION
FRACTION
63 48 47 32
S = SIGN
VARIABLE LENGTH BIT FIELD
P+S P+S-1 P P-1 0
S-1 0
P= STARTING LOCATION ON THE FIELD
S= SIZE OF THE FIELD
CHARACTER STRING & NUMERIC STRING
7 0
A
A= ADDRESS OF FIRST BYTE IN STRING
L = LENGTH OF STRING IN BYTES
7 0
A+L-1
PACKED DECIMAL STRING
7 4 3 0
A
A= ADDRESS OF FIRST BYTE IN STRING
L = NUMBER OF DIGITS IN STRING
A+L-1

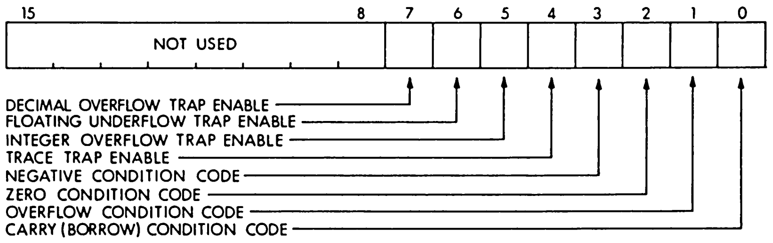
ASSEMBLER NOTATION FOR ADDRESSING MODES

S^#5	forced short literal
#5	optimized short literal
R10	register
(R10)	register deferred
−(R10)	autodecrement
(R10)+	autoincrement
#START	immediate
I^#1	forced immediate
@(R10)+	autoincrement deferred
@#START	absolute
1(R10)	optimized byte displacement
0(R10)	optimized register deferred
@1(R10)	optimized byte displacement deferred
@(R10)	implied byte displacement deferred
START	optimized pc relative
@START	optimized pc relative deferred
1234(R10)	optimized word displacement
@1234(R10)	optimized word displacement deferred
12345678(R10)	longword displacement
@12345678(R10)	longword displacement deferred
B^12(R10)	forced byte displacement
B^START	forced byte pc relative
@B^12(R10)	forced byte displacement deferred
@B^START	forced byte pc relative deferred
W^12(R10)	forced word displacement
W^START	forced word pc relative
@W^12(R10)	forced word displacement deferred
@W^START	forced word pc relative deferred
L^12(R10)	forced longword displacement
L^START	forced longword pc relative
@L^12(R10)	forced longword displacement deferred
@L^START	forced longword pc relative deferred
(R10)[R11]	register deferred indexed
−(R10)[R11]	autodecrement indexed
(R10)+[R11]	autoincrement indexed
@(R10)+[R11]	autoincrement deferred indexed
@#START[R11]	absolute indexed
1(R10)[R11]	optimized byte displacement indexed
0(R10)[R11]	optimized register deferred indexed
@1(R10)[R11]	optimized byte displacement deferred indexed
@(R10)[R11]	implied byte displacement deferred indexed
1234(R10)[R11]	optimized word displacement indexed
@1234(R10)[R11]	optimized word displacement deferred indexed
12345678(R10)[R11]	longword displacement indexed
@12345678(R10)[R11]	longword displacement deferred indexed
START[R11]	optimized pc relative indexed
@START[R11]	optimized pc relative deferred indexed
B^12(R10)[R11]	forced byte displacement indexed
B^START[R11]	forced byte pc relative indexed
@B^12(R10)[R11]	forced byte displacement deferred indexed
@B^START[R11]	forced byte pc relative deferred indexed
W^12(R10)[R11]	forced word displacement indexed
W^START[R11]	forced word pc relative indexed
@W^12(R10)[R11]	forced word displacement deferred indexed
@W^START[R11]	forced word pc relative deferred indexed
L^12(R10)[R11]	forced longword displacement indexed
L^START[R11]	forced longword pc relative indexed
@L^12(R10)[R11]	forced longword displacement deferred indexed
@L^START[R11]	forced longword pc relative deferred

PROCESSOR STATUS LONGWORD



PROCESSOR STATUS WORD


15 8 7 6 5 4 3 2 1 0
NOT USED
DECIMAL OVERFLOW TRAP ENABLE
FLOATING UNDERFLOW TRAP ENABLE
INTEGER OVERFLOW TRAP ENABLE
TRACE TRAP ENABLE
NEGATIVE CONDITION CODE
ZERO CONDITION CODE
OVERFLOW CONDITION CODE
CARRY (BORROW) CONDITION CODE

POWERS OF 2

2^n	n
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
131072	17
262144	18
524288	19
1048576	20
2097152	21
4194304	22
8388608	23
16777216	24

POWERS OF 16

16^n	n
1	0
16	1
256	2
4096	3
65536	4
1048576	5
16777216	6
268435456	7
4294967296	8
68719476736	9
1099511627776	10
17592186044416	11
281474976710656	12
4503599627370496	13
72057594037927936	14
1152921504606846976	15

digital

VAX11 PROGRAMMING CARD

AV-D827A-TE